Class 3 - Styles, Data Templates & Data Binding

Summary:

- Styles
- Controls and Data Templates
- Data Binding

This manual assumes that the resolution of the exercises of Class 2 was successfully implemented up to topic 2.5 (a possible solution is available in the file Aula_02_VS2013.zip).

3.1. Definition of Styles

WPF provides styles that allow you to modify the appearance or behaviour of a control. Styles allow to reuse elements facilitating the creation, coherence and maintenance of graphic interfaces.

In the example of the last class, the labels (`labels`) 'Cursos' ("DETI-Home") and 'Disciplinas/ECTS' ("DETI-Cursos") use the same formatting, so let's define a style to apply in these controls.

Define a style in the `Application Resources` (so that it is available for the entire application) using, for example, the following code:

```
<!-- List header text style -->
<Style x:Key="labelStyle" TargetType="{x:Type Label}">
     <Setter Property="Foreground" Value="White" />
     <Setter Property="VerticalAlignment" Value="Center" />
     <Setter Property="HorizontalAlignment" Value="Left" />
</Style>
```

To apply a style in a control, place the following code in the specific element, removing the local formatting code that is no longer needed.

```
Style="{StaticResource labelStyle}"
```

Apply the style to the three `Labels`. Likewise, define a style for the `Border` used in these `Labels` and apply it in the appropriate places. Use the same formatting that was used in the last class.

Imagine that you want to modify the title "Cursos" on the "DETI-Home" page to use bold. You can inherit properties between styles using the following code when creating the style:

```
BasedOn="{StaticResource baseStyle}"
```

Create a new style that inherits the original properties of the `labelStyle` style but in bold. Apply the style in the appropriate place.

Finally, make the necessary modifications to use the brick color (#9a3324) as the background color in these `Labels`.

3.2. Using Styles

Format the various elements of the page using the xaml code that follows, note that not all have an associated style (`TargetType`), see the difference at the code level.

```
   <!-- Header text style -->
  <Style x:Key="headerTextStyle">
     <Setter Property="Label.VerticalAlignment" Value="Center"></Setter>
     <Setter Property="Label.FontFamily" Value="Trebuchet MS"></Setter>
     <Setter Property="Label.FontWeight" Value="Bold"></Setter>
     <Setter Property="Label.FontSize" Value="18"></Setter>
     <Setter Property="Label.Foreground" Value="#0066cc"></Setter>
  </Style>

<!-- Button style -->
  <Style x:Key="buttonStyle" TargetType="{x:Type Button}">
     <Setter Property="Width" Value="125" />
     <Setter Property="Height" Value="25" />
     <Setter Property="Margin" Value="0,10,0,0" />
     <Setter Property="HorizontalAlignment" Value="Right" />
  </Style>
```

Use these styles to format some elements of the "DETI-Home" page, namely:

- The title of the "DETI-Cursos" page with the style `headerTextStyle;`
- The selection button "Ver" with the style `buttonStyle`.

Modify also the "DETI-Cursos" page to use styles in the title of the page "Disciplinas" with the `headerTextStyle` style.

After these changes, the graphical interface should be similar, but more consistent and facilitating changes. Change the styles associated with the text of the tables (with border) to use and the `Courier New` font with size 14.

3.3. Control Templates

In the previous example, the style is used to define properties. In the case of controls, `Control Templates` can be used to specify how the control itself is designed. Add the following `Template` in the button style and see the result.

```
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type Button}">
        <Grid>
           <Ellipse Fill="#9a3324" Stroke="black"/>
              <ContentPresenter HorizontalAlignment="Center"
  VerticalAlignment="Center"/>
        </Grid>
    </ControlTemplate>
 </Setter.Value>
 </Setter>
```

Apply this template to the "Ver" button. If a shortcut key is associated with the button (with the _), note that to enable this functionality you must activate the option `RecognizesAccessKey="True"` in `ContentPresenter`.

Modify the `Control Template` so that when you go over the button, its color changes to another color of your choice. To do this, after defining the grid with the ellipse, add the following code (note that you have to name the ellipse to be able to identify it).

```
<ControlTemplate.Triggers>
           <Trigger Property="IsMouseOver" Value="True">
             <Setter TargetName="MyEllipse" Property="Fill" Value="Green"/>
        </Trigger>
</ControlTemplate.Triggers>
```

Modify some properties of the `Control Template` (shape, color, trigger) by defining a button as you like. You can use other shapes (rectangle, polygon, ...) colors or triggers.

3.4. Data Binding Examples

Data binding allows to establish the association between a control and data, making it easier to fill in controls with information. In this section we will create the list of courses in a more dynamic way using for such data binding and some associated code.

Start by creating a class that allows to define the list of objects that will fill the ListBox of courses. Add the following class to the code associated with "DETI-Home".

```
public class Curso
{
    private string _nome;
    private int _CodCurso;

    public int CodCurso
    {
        get { return _CodCurso; }
        set { _CodCurso = value; }
    }

    public string Nome
    {
        get { return _nome; }
        set { _nome= value; }
    }
}
```

Add the next class that contains a function that returns the list of courses (to compile you must add `using System.Collections.ObjectModel;`)

```
public class ListaCursos : ObservableCollection <Curso>
{
    public ListaCursos()
    {
        Add(new Curso{Nome ="Computadores e Telemática", CodCurso=8240});
        Add(new Curso{Nome ="Electrónica e Telecomunicações", CodCurso=8204});
        Add(new Curso{Nome ="Engenharia Informática", CodCurso=8295, });
    }
}
```

Now modify the code to make the binding between the courses ListBox and the code now added.

When performing a binding it is necessary to specify:

- A source: which indicates which object or class the binding refers to. Often the source is specified in the data context and is available for various binding.

- A path: if the binding is made to an object, the path (or xPath in the case of XML) indicates which value to indicate in the binding. If this is not specified, binding is done to the entire object.

To define the source, you must specify the DataContext in the xaml through the namespace by adding the following code (you must replace DETI with your namespace).

```
xmlns:local="clr-namespace:DETI"
```

Now you must make the source available by using a key associated with it. You can do this in the Grid.Resources by using the code:

```
<Grid.Resources>
   <local:ListaCursos x:Key="DETICursos"/>
</Grid.Resources>
```

This code allows access to the object within the context of the DETI-Home class. It is possible to bind the information to the ListBox using the following code:

```
   <ListBox Name="cursosListBox" Grid.Column="2" Grid.Row="2"
ItemsSource="{Binding Source={StaticResource DETICursos}}">
```

Run the code and see what happens.

Note that you have to remove or modify the code of the Navigate function as the ListBox now returns an object of type Curso and not a ListBoxItem.

3.5. Data Templates

Similar to the models for the controls (Control Templates), there are models for the data (Data Templates) that allow to modify, for example, the visual aspect of data items in the ListBox and ComboBox controls.

In the previous example, as there is no format associated with the data, Data Binding simply converts the data/objects to String and presents the result.

Add the following code to format the data coming from the Data Source. In this case it is indicated the field to be used to fill the list.

```
<ListBox.ItemTemplate>
   <DataTemplate>
      <Label Content="{Binding Path=Nome}"></Label>
   </DataTemplate>
</ListBox.ItemTemplate>
```

Use a StackPanel to display the name and associated course code in the ListBox.

More information on styles, templates, and data binding can be found at:
https://msdn.microsoft.com/en-us/library/ms745683(v=vs.110).aspx
https://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx

3.6. Functions to the Conversion (optional)

We will now associate a color to each course in the class that was created for this purpose. Associate a string with a hexadecimal value for each course (for example aquamarine #7FFFD4, darksalmon #E9967A and lemonchiffon #FFFACD).

Change the background color of the `ListBoxItem` associated with each course using data binding to associate the color with the background of the labels.

Imagine that you wanted to indicate the name of the color used next to the code, it would be necessary to make a conversion between the hexadecimal string and the name of the color. This can be done with a value converter.

Adapt the following code to make this conversion:

```
    public class ColorConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            // Do the conversion from hexadecimal to string
            return "lemonchiffon";
        }

        public object ConvertBack(object value, Type targetType, object
parameter, CultureInfo culture)
        {
            // Do the conversion from string to hexadecimal
            return null;

        }
      }
```

To compile it is necessary to add the following library:

```
using System.Globalization;
```

The use of the conversion function in xaml is indicated within the binding as follows:

```
Converter={StaticResource myConverter}
```

It is necessary to create an instance of the conversion class in resources (for example Grid.Resources) with the following code:

```
<Grid.Resources>
    <local:ColorConverter x:Key="myConverter"/>
</Grid.Resources>
```